Homework 1

Barak-Nadav Diker

December 23, 2024

Contents

1 Finding the Bias Scale-Factor miss align errors

Our model of errors is

$$\tilde{f} = (I_3 + S_a + M_a)f + b_a + w_a \tag{1}$$

Where $S_a \in diag_{3\times 3}\mathbb{R}$ and $M_a \in AntiSymm_{3\times 3}\mathbb{R}$ and $b_a \in \mathbb{R}^3$

Note the following derivation

$$\tilde{f} - f = (S_a + M_a)f + b_a$$

Concating the following matrices would generate the following matrix

$$M_{3\times 4} = \left[\ M_a + S_a \ \middle|\ b_a\ \right]$$

Given that $M_{3\times 4}$ matrix we can simplify equation 1 to be

$$\tilde{f} - f_{3\times 1} = M_{3\times 4}\begin{pmatrix} f_{3\times 1} \\ 1 \end{pmatrix}$$

For example, Estimating left column of M with down x direction measurements

$$\tilde{f}^x_{down} - \begin{pmatrix} -g \\ 0 \\ 0 \end{pmatrix} = M \begin{pmatrix} -g \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

We can concate all the vector equations together and we'll infer the following matrix equation

$$\begin{pmatrix} \tilde{f}^x_{down} & \tilde{f}^{up}_x & \cdots & \tilde{f}^z_{up} \end{pmatrix} - \begin{pmatrix} -g & g & 0 & 0 & 0 & 0 \\ 0 & 0 & -g & g & 0 & 0 \\ 0 & 0 & 0 & 0 & -g & g \end{pmatrix} = M \begin{pmatrix} -g & g & 0 & 0 & 0 & 0 \\ 0 & 0 & -g & g & 0 & 0 \\ 0 & 0 & 0 & 0 & -g & g \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

I'll denote the matrix $A \in \mathbb{R}^{4\times 6}$ like so

$$A_{4\times 6} = \begin{pmatrix} -g & g & 0 & 0 & 0 & 0 \\ 0 & 0 & -g & g & 0 & 0 \\ 0 & 0 & 0 & 0 & -g & g \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

and

$$z_{3\times 6} = M_{3\times 4} A_{4\times 6}$$

$$z_{3\times 6} A^T (AA^T)^{-1} = M_{3\times 4} A_{4\times 6} A^T (AA^T)^{-1}$$

$$z_{3\times 6} A^T (AA^T)^{-1} = M_{3\times 4} \tag{2}$$

Equation 2 gives us the best M matrix for a given measurement so our estimator is an Unbiased estimator, so for simple calculation we'll have

$$\hat{M} = \frac{1}{n} \sum_{i=1}^{n} M = \frac{1}{n} \sum_{i=1}^{n} z_{3\times 6} A^T (AA^T)^{-1} \tag{3}$$

2 Implementation

Here I am calculating the Matrix A of equation 2 using the Following code

```
# Calculating A
import numpy as np
#g = 9.81 #m/s^2
g =1
D = np.zeros((3,6))
D[0,0] = -g ; D[0,1] = g ; D[1,2] = -g ; D[1,3] = g ; D[2,4] =-g ;
↪  D[2,5]=g
E = np.ones((1,6),dtype=float)
print("A=")
print(np.vstack((D,E)))
```

```
A=
[[-1.  1.  0.  0.  0.  0.]
 [ 0.  0. -1.  1.  0.  0.]
 [ 0.  0.  0.  0. -1.  1.]
 [ 1.  1.  1.  1.  1.  1.]]
```

$$\begin{matrix} -9.81 & 9.81 & 0. & 0. & 0. & 0. \\ 0. & 0. & -9.81 & 9.81 & 0. & 0. \\ 0. & 0. & 0. & 0. & -9.81 & 9.81 \\ 1. & 1. & 1. & 1. & 1. & 1. \end{matrix}$$

$$\begin{matrix} -1. & 1. & 0. & 0. & 0. & 0. \\ 0. & 0. & -1. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & -1. & 1. \\ 1. & 1. & 1. & 1. & 1. & 1 \end{matrix}$$

2.1 Core implementation

Here I am finding the errors as I have developed in the theory section

In the end I'll obtain the matrix $M \in \mathbb{R}^{3\times 4}$ which consist of b_a, S_a, M_a

```
import numpy as np
import pandas as pd
x_mi = pd.read_csv("./dataset_work/sec300xminus.csv")
y_mi = pd.read_csv("./dataset_work/sec300yminus.csv")
z_mi = pd.read_csv("./dataset_work/sec300zminus.csv")
x_pl = pd.read_csv("./dataset_work/sec300xplus.csv")
y_pl = pd.read_csv("./dataset_work/sec300yplus.csv")
z_pl = pd.read_csv("./dataset_work/sec300zplus.csv")
x_mi = x_mi[['gFx','gFy','gFz']].to_numpy()
```

```
y_mi = y_mi[['gFx','gFy','gFz']].to_numpy()
z_mi = z_mi[['gFx','gFy','gFz']].to_numpy()
x_pl = x_pl[['gFx','gFy','gFz']].to_numpy()
y_pl = y_pl[['gFx','gFy','gFz']].to_numpy()
z_pl = z_pl[['gFx','gFy','gFz']].to_numpy()
A = np.array(A)

x_mi -= np.array([-1,0,0])
y_mi -= np.array([0,-1,0])
z_mi -= np.array([0,0,-1])
x_pl -= np.array([1,0,0])
y_pl -= np.array([0,1,0])
z_pl -= np.array([0,0,1])


n=0
sum_of_M = np.zeros((3,4))
for x_m,y_m,z_m,x_p,y_p,z_p in zip(x_mi,y_mi,z_mi,x_pl,y_pl,z_pl):
    n+=1
    x_m = np.expand_dims(x_m,1).T
    y_m = np.expand_dims(y_m,1).T
    z_m = np.expand_dims(z_m,1).T
    x_p = np.expand_dims(x_p,1).T
    y_p = np.expand_dims(y_p,1).T
    z_p = np.expand_dims(z_p,1).T
    z= np.concatenate((x_m,x_p,y_m,y_p,z_m,z_p)).T
    M  = z @ np.transpose(A) @ np.linalg.inv(A@np.transpose(A))
    sum_of_M += M
    #break
esti_M = sum_of_M/n #, sum_of_M.shape
print("The Estimated M which is unbiased estimator is \n",esti_M)
print("The S_a error is \n",
↪  np.diag((esti_M[0,0],esti_M[1,1],esti_M[2,2])))
M_a = esti_M.copy()
M_a = M_a[0:3,0:3]
M_a[0,0] = 0
M_a[1,1] = 0
M_a[2,2] = 0
print("The M_a error is \n", M_a)
print("The Bias of IMU is \n", esti_M[:,3])

#return sum_of_M.shape
#return sum_of_M/n #, sum_of_M.shape
#return  x_minus[['gFx','gFy','gFz']].to_numpy().shape[0]
#return x_minus[['gFx','gFy','gFz']].to_numpy().shape
```

```
The Estimated M which is unbiased estimator is
 [[-0.02861075 -0.00391678 -0.01046105 -0.00122523]
```

```
 [ 0.00165313 -0.00751652 -0.00525399  0.0009181 ]
 [ 0.02379594 -0.02749688 -0.00575567  0.13867218]]
The S_a error is
 [[-0.02861075  0.          0.        ]
 [ 0.         -0.00751652  0.        ]
 [ 0.          0.         -0.00575567]]
The M_a error is
 [[ 0.         -0.00391678 -0.01046105]
 [ 0.00165313  0.         -0.00525399]
 [ 0.02379594 -0.02749688  0.        ]]
The Bias of IMU is
 [-0.00122523  0.0009181   0.13867218]
```

3 2.6 Only bias error

In this exercise, I take only one accelerometer and consider only the bias error

for this derivation I'll consider the following equation 1

$$\tilde{f} = (I_3 + S_a + M_a)f + b_a + w_a$$

And if I am assuming only bias error, I'll have the following equation

$$\tilde{f} = f + b_a \tag{4}$$

and for estimating equation 4 , consider

$$b_a = \tilde{f} - f \tag{5}$$

which is an "unbiased estimator"

$$\hat{b}_a = \frac{1}{n}\sum_{i=0}^{n} b_a = \frac{1}{n}\sum_{i=0}^{n}(\tilde{f} - f) \tag{6}$$

```python
import numpy as np
import pandas as pd
x_mi = pd.read_csv("./dataset_work/sec300xminus.csv")
x_mi = x_mi[['gFx','gFy','gFz']].to_numpy()
diff_x = x_mi-np.array([-1 , 0, 0])
print("The bias of the accelerometer using average is \n")
print(diff_x.sum(axis=0) / diff_x.shape[0])
print(diff_x.shape[0])
```

```
The bias of the accelerometer using average is

[ 0.02406121 -0.00294848  0.23269958]
149878
```

4 2.7 calculate the bias errors using only average

In order to do so I'll Implement 2.6 for all accelerometer

```
import numpy as np
import pandas as pd
def estimate_only_bias(address=
↪  "./dataset_work/sec300xminus.csv",true_force = [-1,0,0]): # true_force
↪  = [-1,0,0]
    x_mi = pd.read_csv(address)
    x_mi = x_mi[['gFx','gFy','gFz']].to_numpy()
    diff_x = x_mi-np.array(true_force)
    return diff_x.sum(axis=0) / diff_x.shape[0]

bias_term = estimate_only_bias("./dataset_work/sec300xminus.csv",[-1,0,0])
bias_term +=
↪  estimate_only_bias("./dataset_work/sec300yminus.csv",[0,-1,0])
bias_term +=
↪  estimate_only_bias("./dataset_work/sec300zminus.csv",[0,0,-1])
bias_term += estimate_only_bias("./dataset_work/sec300xplus.csv",[1,0,0])
bias_term += estimate_only_bias("./dataset_work/sec300yplus.csv",[0,1,0])
bias_term += estimate_only_bias("./dataset_work/sec300zplus.csv",[0,0,1])
print("The bias error using only averages is ")
print(bias_term/6)
```

```
The bias error using only averages is
[-0.00119208  0.00093832  0.13886594]
```

4.1 Special note

using the Entire error model the bias is very close to the average bias

- The Bias of IMU using entire model is [-0.00122523 0.0009181 0.13867218]
- The Bias of IMU using only average is [-0.00119208 0.00093832 0.13886594]

4.2 Important conclusions

Given our entire error model lets see how well the error model fits the data by plotting the error which will be defined by $|f - \tilde{f}|_2$ lets see how It'll look like

our calculated $M_{3\times4}$ matrix is

```
import numpy as np
import sys
import pandas as pd
```

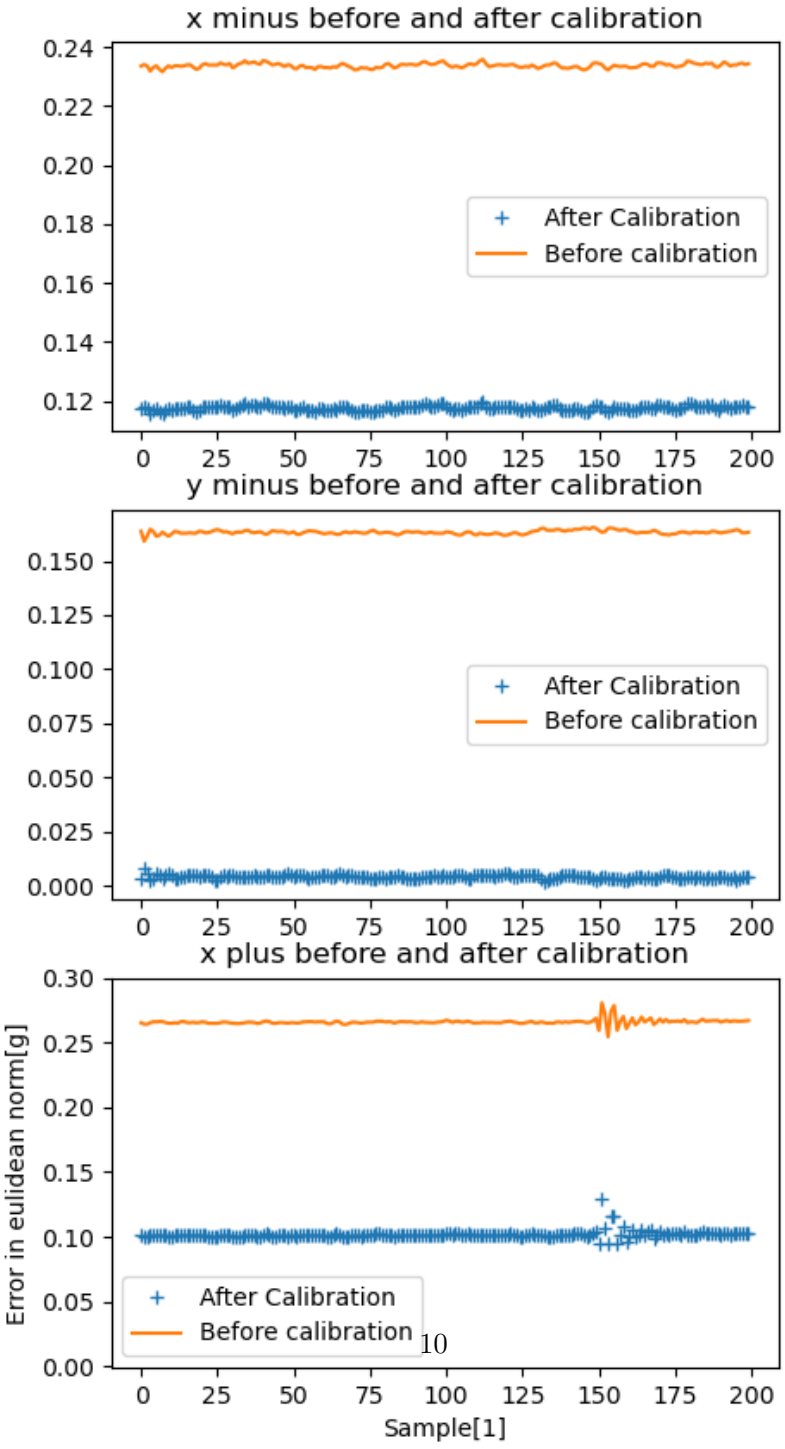
-0.02861075	-0.00391678	-0.01046105	-0.00122523
0.00165313	-0.00751652	-0.00525399	0.0009181
0.02379594	-0.02749688	-0.00575567	0.13867218

```
from matplotlib import pyplot as plt
x_mi = pd.read_csv("./dataset_work/sec300xminus.csv")
y_mi = pd.read_csv("./dataset_work/sec300yminus.csv")
z_mi = pd.read_csv("./dataset_work/sec300zminus.csv")
x_pl = pd.read_csv("./dataset_work/sec300xplus.csv")
y_pl = pd.read_csv("./dataset_work/sec300yplus.csv")
z_pl = pd.read_csv("./dataset_work/sec300zplus.csv")
x_mi = x_mi[['gFx','gFy','gFz']].to_numpy()
y_mi = y_mi[['gFx','gFy','gFz']].to_numpy()
z_mi = z_mi[['gFx','gFy','gFz']].to_numpy()
x_pl = x_pl[['gFx','gFy','gFz']].to_numpy()
y_pl = y_pl[['gFx','gFy','gFz']].to_numpy()
z_pl = z_pl[['gFx','gFy','gFz']].to_numpy()
#A = np.array(A)
M = np.array(M)
M = M + np.concatenate([ np.identity(3) , np.array([[0,0,0]]).T],axis=1)
x_mi_improved =x_mi -  M @ np.array([-1,0,0,1])
x_mi_usual =x_mi -  np.array([-1,0,0])
y_mi_improved =y_mi -   M@np.array([0,-1,0,1])
y_mi_usual =y_mi -  np.array([0,-1,0])
z_mi -=M@np.array([0,0,-1,1])
#x_pl -=M@np.array([1,0,0,1])
x_pl_improved = x_pl - M @np.array([1,0,0,1])
x_pl_usual = x_pl - np.array([1,0,0])
#print(np.linalg.norm(x_pl_usual[30]))
y_pl -=M@np.array([0,1,0,1])
z_pl -=M@np.array([0,0,1,1])

x_mi_improved = np.array([np.linalg.norm(v) for v in
↪  x_mi_improved]).reshape(-1)
x_mi_usual = np.array([np.linalg.norm(v) for v in x_mi_usual]).reshape(-1)

y_mi_improved = np.array([np.linalg.norm(v) for v in
↪  y_mi_improved]).reshape(-1)
y_mi_usual = np.array([np.linalg.norm(v) for v in y_mi_usual]).reshape(-1)
x_pl_improved = np.array([np.linalg.norm(v) for v in
↪  x_pl_improved]).reshape(-1)
x_pl_usual = np.array([np.linalg.norm(v) for v in x_pl_usual]).reshape(-1)
#print(x_mi[:3])
fig, ax = plt.subplots(3,1,figsize=(5,10))
ax[0].set_title("x minus before and after calibration")
ax[0].plot(np.arange(len(x_mi[:200])),x_mi_improved[:200],'+',label='After
↪  Calibration')
```

```
ax[0].plot(np.arange(len(x_mi[:200])),x_mi_usual[:200],label='Before
↪  calibration')
ax[0].legend()
ax[1].set_title("y minus before and after calibration")
ax[1].plot(np.arange(len(y_mi[:200])),y_mi_improved[:200],'+',label='After
↪  Calibration')
ax[1].plot(np.arange(len(y_mi[:200])),y_mi_usual[:200],label='Before
↪  calibration')
ax[1].legend()
ax[2].set_title("x plus before and after calibration")
ax[2].plot(np.arange(len(x_pl[:200])),x_pl_improved[:200],'+',label='After
↪  Calibration')
ax[2].plot(np.arange(len(x_pl[:200])),x_pl_usual[:200],label='Before
↪  calibration')
ax[2].legend()
plt.xlabel("Sample[1]")
plt.ylabel("Error in eulidean norm[g]")
plt.ylim(-0.001,0.3)
plt.savefig('error_fixed_unfixed.png')
#plt.show()
```


x minus before and after calibration
0.24
0.22
0.20
0.18
0.16
0.14
0.12
After Calibration
Before calibration
0
25
50
75
100
125
150
175
200
y minus before and after calibration
0.150
0.125
0.100
0.075
0.050
0.025
0.000
After Calibration
Before calibration
0
25
50
75
100
125
150
175
200
x plus before and after calibration
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Error in eulidean norm[g]
After Calibration
Before calibration
0
25
50
75
100
125
150
175
200
Sample[1]